Swapping Lemmas for Regular and Context-Free Languages

Tomoyuki Yamakami

School of Computer Science and Engineering, University of Aizu
90 Kami-Iawase, Tsuruga, Ikki-machi, Aizu-Wakamatsu
Fukushima 965-8580, Japan

Abstract. In formal language theory, one of the most fundamental tools, known as pumping lemmas, is extremely useful for regular and context-free languages. However, there are natural properties for which the pumping lemmas are of little use. One of such examples concerns a notion of advice, which depends only on the size of an underlying input. A standard pumping lemma encounters difficulty in proving that a given language is not regular in the presence of advice. We develop its substitution, called a swapping lemma for regular languages, to demonstrate the non-regularity of a target language with advice. For context-free languages, we also present a similar form of swapping lemma, which serves as a technical tool to show that certain languages are not context-free with advice.

Keywords: regular language, context-free language, pushdown automaton, advice, pumping lemma, swapping lemma

1 Motivation and Overview

Since their creation in the 1940's, formal languages and grammars have found many applications in, for instance, programming languages and artificial intelligence. A rapid development of the Internet, in particular, may treat one-way finite automata as a meaningful model of hand-held communication devices with a small amount of fast cache memory, which can process incoming information streamlined through a communication channel. Such types of automata are closely associated with the classical notions of *regular languages* and *context-free languages*, which were also classified respectively as type 3 and type 2 by Chomsky. These notions, which are foundations to formal language theory, have been a key subject of many core undergraduate curricula in computer science. The beauty of these notions comes from their simplicity and applicability.

One of the most useful tools in formal language theory is so-called *pumping lemmas* [1] for regular and context-free languages. Most undergraduate textbooks in this classic theory describe these lemmas (and their variants) as a powerful but fundamental tool. A standard methodology of proving the non-regularity of a target language, for instance, is to apply the pumping lemma to the language, deriving a desired contradiction if we begin with assuming the regularity of the language (see, *e.g.*, [4] for its argument). For certain natural properties of languages, however, the pumping lemmas are not in the most useful form and we thus need to develop another form of lemmas to prove those properties. A typical example concerns "advised computation."

One way of enhancing the power of machine's language recognition is to provide a piece of *advice*, which is a string depending only on the length of input strings, to the original input strings. Karp and Lipton [5] formulated a mechanism of such advice for polynomial time-bounded computation. Inspired by the work of Karp and Lipton, Damm and Holzer [3] and Tadaki, Yamakami, and Lin [6] discussed the computational power of finite automata when advice is provided as supplemental information to their single read-only input tapes. See Section 2 for the formal definition of automata that take advice. As in [6], we use the notation REG/n to denote the family of languages recognized by deterministic finite automata if advice of size exactly n is provided *in parallel* to an input of size n. Similarly, we define CFL/n for the language family characterized by nondeterministic pushdown automata with such advice.

As is known, the advised family REG/n is quite different from REG, the family of regular languages. Typical context-sensitive languages, such as $L_{3eq} = \{a^n b^n c^n \mid n \geq 0\}$, naturally fall into this advised family REG/n. On the contrary, certain deterministic context-free languages, such as $Equal = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$ and $GT = \{w \in \{0,1\}^* \mid \#_0(w) > \#_1(w)\}$, where $\#_a(w)$ denotes the number of a's in w, do not belong to REG/n. *How can we prove this fact?* Now, let us try to prove that, for example, *Equal* is not in REG/n using the pumping lemma for regular languages. Assume otherwise that *Equal* is characterized

by a certain regular language L with advice strings over an alphabet Γ. Now, we apply the pumping lemma by picking a pumping-lemma constant m and then choosing an appropriate string w (together with an advice string) in L of length at least m. Using its decomposition $w = xyz$, the pumping lemma pumps this chosen string w with advice given in parallel to w, generating a new series of strings of the form xy^iz. These pumped strings must belong to L; however, are they still in *Equal* with the appropriate advice strings? At this point, we encounter a serious flaw in our argument. The pumping process unwisely pumps the original string as well as the valid advice string. Since the pumping lemma forces the size of pumped strings to change, their associated pumped advice might no longer be "valid" advice for *Equal*. Therefore, we cannot conclude that the pumped strings are actually in *Equal* with advice. To avoid this pitfall, we need to develop a new lemma, which keeps advice valid before and after the application of the lemma.

In this paper, we shall present such a desired lemma, which we refer to as the *swapping lemma*, encompassing an essential nature of regular languages. In many cases, this new lemma is as powerful as the pumping lemma is. As examples, we shall later demonstrate, by a direct application of the swapping lemma, that the context-free language $Pal = \{ww^R \mid w \in \{0,1\}^*\}$ (even-length palindromes), where w^R is w in reverse, and the aforementioned languages *Equal* and *GT* cannot be in REG/n (and therefore, they are not in REG). The last two examples show a separation of DCFL, the family of deterministic context-free languages, from the advised class REG/n. This immediately yields the class separation DCFL/$n \neq$ REG/n, which has not been known so far. Our proof of the swapping lemma for regular languages is considerably simple and can be obtained from a direct application of the pigeonhole principle.

Likewise, we also introduce a similar form of swapping lemma for context-free languages to deal with the non-membership to the advised family CFL/n. With help of this swapping lemma, as an example, we prove that the language $Dup = \{ww \mid w \in \{0,1\}^*\}$ (duplicating strings) is not in CFL/n (and therefore not in CFL, the family of context-free languages). Another (slightly contrived) example is the language $Equal_6$ consisting of all strings w over an alphabet of 6 symbols together with a special separator # for which each symbol except # appears the same number of times in w. Since $Equal_6$ is in the complement of CFL, denoted co-CFL, we obtain a strong separation between CFL/n and co-CFL/n; in other words, CFL/n is not closed under complementation. Our proof of the swapping lemma for context-free languages is quite different from a standard proof of the pumping lemma for context-free languages. Rather than using context-free grammars, our proof deals with a certain restricted form of nondeterministic pushdown automata to track down their behaviors in terms of transitions of first-in last-out stack contents.

The main purposes of this paper are summarized as follows: (i) to introduce the two swapping lemmas for regular and context-free languages, (ii) to give their proofs by exploiting certain structural properties of finite automata, and (iii) to demonstrate the strong separations between DCFL/n and REG/n and between CFL/n and co-CFL/n.

We hope that the results of this paper should contribute to a fundamental progress of formal language theory and revive fresh interest in basic notions of regular languages and context-free languages.

2 Notions and Notation

The *natural numbers* are nonnegative integers and we write $\mathbb{N}$ to denote the set of all such numbers. For any two integers m, n with $m \leq n$, the notation $[m,n]_{\mathbb{Z}}$ stands for the integer interval $\{m, m+1, m+2, \ldots, n\}$.

An *alphabet* is a nonempty finite set and our alphabet is denoted by either Σ or Γ. A *string* over an alphabet Σ is a series of symbols from Σ. In particular, the *empty string* is always denoted λ. The notation Σ^* expresses the set of all strings over Σ. The *length* of a string w, denoted $|w|$, is the total number of symbols in w. For each length $n \in \mathbb{N}$, we write Σ^n (resp., $\Sigma^{\leq n}$) for the set of all strings over Σ of length exactly n (resp., at most n). For any non-empty string w and any number $i \in [0, |w|]_{\mathbb{Z}}$, $pref_i(w)$ denotes the first i symbols of w; namely, the substring s of w such that $|s| = i$ and $sx = w$ for a certain string x. In particular, $pref_0(w) = \lambda$ and $pref_{|w|}(w) = w$. Similarly, let $suf_i(w)$ be the last i symbols of w. For any string x of length n and two arbitrary indices $i, j \in [1,n]_{\mathbb{Z}}$ with $i \leq j$, the notation $midd_{i,j}(x)$ denotes the string obtained from x by deleting the first i symbols and the last $n-j$ symbols of x; thus, $midd_{i,j}(x)$ contains exactly $j-i$ symbols taken from x. As a special case, $midd_{i,i}(x)$ expresses the ith symbol of x. It always holds that $x = pref_i(x)midd_{i,j}suf_{n-j}(x)$.

For any language L over Σ, the *complement* $\Sigma^* - L$ of L is denoted $\overline{L}$ whenever Σ is clear from the context.

The *complement* of a family $\mathcal{C}$ of languages is the collection of all languages whose complements are in $\mathcal{C}$. We use the conventional notation co-$\mathcal{C}$ to describe the complement of $\mathcal{C}$.

We denote by REG the family of *regular languages*. Similarly, the notation CFL represents the family of *context-free languages*. We also use the notation DCFL for the *deterministic context-free language* family. We assume the reader's familiarity with fundamental mechanisms of *one-tape one-head finite-state automata* and their variant, *pushdown automata* (see, *e.g.*, [4] for their formal definitions). Concerning these machine models, REG, CFL, and DCFL can be characterized by deterministic automata (or dfa's), nondeterministic pushdown automata (or npda's), and deterministic pushdown automata (or dpda's), respectively.

Now, let us briefly state the notion of *advice* in the form given in [6], which is slightly different from [3]. First, we explain how to provide advice strings to finite automata using a "track" notation. Consider a finite automaton with one scanning head moving on a read-only input tape, which consists of tape cells indexed with integers. For simplicity, the leftmost symbol of any input string is always placed in the cell indexed 1. Now, we split the tape into two tracks. The upper track contains an original input x given to the machine and the lower track carries a piece of advice, which is a string w (over a possibly different alphabet) of the length $|x|$. More precisely, the tape contains n tape cells consisting of the string $\left[\begin{smallmatrix} x \\ w \end{smallmatrix}\right] = \left[\begin{smallmatrix} x_1 \\ \sigma_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} x_2 \\ \sigma_2 \end{smallmatrix}\right]\cdots\left[\begin{smallmatrix} x_n \\ \sigma_n \end{smallmatrix}\right]$, where $x = x_1x_2\cdots x_n$ and $w = \sigma_1\sigma_2\cdots\sigma_n$, in such a way that each ith cell contains the symbol $\left[\begin{smallmatrix} x_i \\ \sigma_i \end{smallmatrix}\right]$. The machine takes advantages of this advice w to determine whether it accepts the input x or not. To deal with all different lengths, advice is generally given as a form of function* h (which is called an *advice function*) mapping $\mathbb{N}$ to Γ^*, where Γ is another alphabet, such that $|h(n)| = n$ for any length $n \in \mathbb{N}$.

The succinct notation REG/n, given in [6], denotes the collection of all languages L over an alphabet Σ such that there are an advice function h and a dfa M for which, for all strings $x \in \Sigma^*$, $x \in L$ iff M accepts $\left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right]$. Since each dfa M characterizes a certain regular language, say, L', the above definition of REG/n can be made machine-independent by replacing the dfa M by the regular language L': let $L \in$ REG/n if there exist an advice function h and another language L' in REG for which, for all strings $x \in \Sigma^*$, $x \in L$ iff $\left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right] \in L'$. Similarly, we can define the advised families CFL/n and DCFL/n from CFL and DCFL, respectively.

To help the reader grasp the concept of advice, we shall see a quick example of how to prepare such advice and use it to accept our target strings. Consider the context-sensitive language $L_{3eq} = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. It is obvious that L_{3eq} is not a regular language. Let us claim that L_{3eq} belongs to REG/n.

Example 2.1 Consider the non-regular language $L_{3eq} = \{a^nb^nc^n \mid n \in \mathbb{N}\}$. It is easy to check that L_{3eq} belongs to REG/n by choosing an advice function h defined as $h(n) = a^{n/3}b^{n/3}c^{n/3}$ if $n \equiv 0 \pmod 3$ and $h(n) = 0^n$ otherwise. How can we recognize this language with advice? Consider a dfa M that behaves as follows. On input $\left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right]$ with advice $h(|x|)$, if $x = \lambda$, then accept the input immediately. Otherwise, check if the first bit of $h(n)$ is a. If so, check if $x = h(|x|)$ by moving the tape head one by one to the right. This is possible by scanning the upper and lower tracks at once at each cell. If the first bit of $h(n)$ is c instead, reject the input. It is obvious that M accepts $\left[\begin{smallmatrix} x \\ h(|x|) \end{smallmatrix}\right]$ iff $x \in L_{3eq}$. Therefore, L_{3eq} belongs to REG/n. □

Another example is a context-free language $Pal = \{ww^R \mid w \in \{0,1\}^*\}$ (even-length palindromes), where w^R denotes w in reverse. It is well-known that Pal is located outside DCFL; however, as we show below, advice helps Pal sit inside DCFL/n.

Example 2.2 The non-"deterministic context-free" language Pal belongs to DCFL/n. This claim is shown as follows. It is well-known that the "marked" language $Pal_{\#} = \{w\#w^R \mid w \in \{0,1\}^*\}$, where $\#$ is a center marker, can be recognized by a certain dpda, say, M (see, *e.g.*, [4] for its proof). The center marker in $Pal_{\#}$ gives M a cue to switch the dpda's inner mode at the time the dpda's head moves from w to w^R. More precisely, the dpda M stores the left substring w in its stack and, upon its cue from the marker, M checks whether this stack content matches the rest of the tape content. Since there is no center marker in Pal, we instead use an advice function h to mark the boundary between w and w^R in ww^R. We define $h(0) = \lambda$, $h(n) = 0^{n/2-1}101^{n/2-1}$ if n is even with $n \geq 2$, and $h(n) = 1^n$ if n is odd. The first occurrence of 10 in $h(n)$

*In the original definition of advice functions by Karp and Lipton [5], these functions are not necessarily computable. This is mainly because we are interested in how much "information" with which each piece of advice provides an underlying machine, rather than how to generate such information.

in the even case, for instance, signals the time of transition from w to w^R in a similar way as the dpda M does for $Pal_{\#}$. This advice places Pal in DCFL/n. □

3 Swapping Lemma for Regular Languages

Our goal of this section is to develop a new form of useful lemma, which can substitute the well-known *pumping lemma* [1] for regular languages, even in the presence of advice. We have seen the power of advice in Examples 2.1 and 2.2: advice helps dfa's recognize non-"regular" languages and also helps dpda's recognize non-"deterministic context-free" languages. When we want to show that a certain language L, such as $Equal$ and Pal, does not belong to REG/n, a standard way (stated in many undergraduate textbooks) is an application of the pumping lemma for regular languages. A basic scheme of the standard pumping lemma (and many of its variants) states that, for any infinite regular language L and any string w in L, as far as its string size is large enough (at least a certain constant that depends only on L), we can always pump the string w (by repeating a middle portion of w) while keeping its pumped string within the language L. Unfortunately, as discussed in Section 1, the pumping lemma is not as useful as we hope it should be, when we wish to prove that certain languages are located outside the advised family REG/n. To achieve our goal, we need to develop a different type of lemma, which we call the *swapping lemma* for regular languages.

We begin with a simpler form of our swapping lemma.

Lemma 3.1 [Swapping Lemma for Regular Languages] *Let L be any infinite regular language over an alphabet Σ with $|\Sigma| \geq 2$. There exists a positive integer m (called a swapping-lemma constant) such that, for any integer $n \geq 1$ and any subset S of $L \cap \Sigma^n$ of cardinality more than m, the following condition holds: for any integer $i \in [0,n]_{\mathbb{Z}}$, there exist two strings $x = x_1x_2$ and $y = y_1y_2$ in S with $|x_1| = |y_1| = i$ and $|x_2| = |y_2|$ satisfying that (i) $x \neq y$, (ii) $y_1x_2 \in L$, and (iii) $x_1y_2 \in L$.*

Proof. We prove the lemma by a simple counting argument with use of the pigeonhole principle. Let L be any infinite regular language over an alphabet Σ. Choose a dfa $M = (Q, \Sigma, \delta, q_0, F)$ that *recognizes* L, where Q is a finite set of inner states, δ is a transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. We define our swapping-lemma constant m as $|Q|$. Let n be any integer at least 1 and let S be any subset of $L \cap \Sigma^n$ with $|S| > m$. Clearly, $|S| \geq 2$. Choose an arbitrary index $i \in [0,n]_{\mathbb{Z}}$. If either $i = 0$ or $i = n$, then the lemma is trivially true (by choosing any two distinct strings x, y in S). Henceforth, we assume that $n \geq 2$ and $1 \leq i \leq n-1$. Consider internal states just after scanning the ith cell. Since $|S| > |Q|$, there are two distinct strings $x, y \in S$ for which M enters the same internal state, say q, after reading the ith symbol of x as well as y. Since the dfa cannot distinguish $pref_i(x)$ and $pref_i(y)$ after reading these substrings, M should accept the swapped strings $pref_i(x)suf_{n-i}(y)$ and $pref_i(y)suf_{n-i}(x)$. This completes the proof. □

Notice that, in the no-advice case, our swapping lemma can be used as a substitute for the pumping lemma when a target language L is not "slim" enough (*i.e.*, $|L \cap \Sigma^n| > m$). Let us demonstrate two simple examples of how to use our swapping lemma. The first example is the context-free language $Pal = \{ww^R \mid w \in \{0,1\}^*\}$.

Example 3.2 The context-free language Pal is not in REG/n (and thus not in REG). Assume that Pal belongs to REG/n and apply the swapping lemma for regular languages. Since $Pal \in$ REG/n, there are a language $L \in$ REG over an alphabet Σ and an advice function h such that, for every string $x \in \{0,1\}^*$, $[\ {}^{x}_{h(|x|)}\ ] \in L$ iff $x \in Pal$. Take a swapping-lemma constant m that satisfies the swapping lemma for L. Choose $n = 2m$ and $i = n/2$. Let a subset S of $L \cap \Sigma^n$ be $S = \left\{[\ {}^{x}_{h(n)}\ ] \in L \mid |x| = n\right\}$. Notice that $|S| \geq 2^{n/2} > m$. By the lemma, there are two distinct strings $x, y \in \{0,1\}^n$ that force $[\ {}^{x}_{h(n)}\ ]$ and $[\ {}^{y}_{h(n)}\ ]$ to fall into S. By letting $u_1 = pref_i(x)$ and $u_2 = pref_i(y)$, the strings x and y are written as $x = u_1u_1^R$ and $y = u_2u_2^R$. Now, let us consider the two swapped strings $u_1u_2^R = pref_{n/2}(x)suf_{n/2}(y)$ and $u_2u_1^R = pref_{n/2}(y)suf_{n/2}(x)$. These strings are clearly not of the form ww^R, and thus the swapped strings $[\ {}^{u_1u_2^R}_{h(n)}\ ]$ and $[\ {}^{u_2u_1^R}_{h(n)}\ ]$ cannot belong to L. This is a contradiction against the swapping lemma. Therefore, Pal is not in REG/n. □

The use of the subset S in the swapping lemma, Lemma 3.1, is of great importance in dealing with the advised family REG/n because, for instance, S in the above example cannot be defined as $S = L \cap \Sigma^n$ in

order to lead to a desired contradiction. There are also cases that require more dexterous choices of S. One of those cases is the non-regular language $Equal = \{w \in \{0,1\}^* \mid \#_0(w) = \#_1(w)\}$.

Example 3.3 The deterministic context-free language *Equal* is not in REG/n. This statement was first stated in [6]. Our purpose here is to apply our swapping lemma to reprove this known result. Assume that *Equal* is in REG/n. There are a regular language L and an advice function h such that, for every binary string x, $x \in Equal$ iff $[\ {}^{x}_{h(|x|)}\ ] \in L$. Take a swapping-lemma constant m and set $n = 2m$ as well as $i = n/2$. In this example, we cannot take a subset $S = \left\{[\ {}^{x}_{h(n)}\ ] \in L \mid |x| = n\right\}$ as we have done in Example 3.2; instead, we rather choose $n/2+1$ distinct strings $w_0, w_1, w_2, \ldots, w_{n/2} \in \{0,1\}^n$, where $w_k = 0^k 1^{n/2-k} 0^{n/2-k} 1^k$ for each index $k \in [0, n/2]_{\mathbb{Z}}$, and we then define $S = \left\{[\ {}^{w_0}_{h(n)}\ ], \ldots, [\ {}^{w_{n/2}}_{h(n)}\ ]\right\}$. Clearly, the cardinality $|S|$ is more than m. The crucial point of the choice of w_k's is explained as $\#_0(pref_{n/2}(w_k)) = k$ for any number $k \in [0, n/2]_{\mathbb{Z}}$. The swapping lemma provides two distinct strings $x = w_j$ and $y = w_k$ ($j \neq k$) such that $[\ {}^{x}_{h(n)}\ ], [\ {}^{y}_{h(n)}\ ] \in S$ and $[\ {}^{u_1}_{h(n)}\ ], [\ {}^{u_2}_{h(n)}\ ] \in L$, where the swapped strings u_1 and u_2 are of the form $u_1 = pref_{n/2}(x)suf_{n/2}(y)$ and $u_2 = pref_{n/2}(y)suf_{n/2}(x)$. It easily follows that

$$\#_0(u_1) = \#_0(pref_{n/2}(x)) + \#_0(suf_{n/2}(y)) = j + \frac{n}{2} - k \neq \frac{n}{2}$$

since $j \neq k$. This contradicts the result that $[\ {}^{u_1}_{h(n)}\ ] \in L$. Therefore, *Equal* cannot be in REG/n. □

Next, we present a more general form of our swapping lemma. In Lemma 3.1, two strings x and y are both split into two blocks and one of these blocks is used for swapping. In the next lemma, in contrast, these strings are split into any fixed number of blocks, one of which is actually used for swapping. This form is useful when we want to show that, for instance, the non-regular language $GT = \{w \in \{0,1\}^* \mid \#_0(w) > \#_1(w)\}$ does not belong to REG/n.

Lemma 3.4 [Swapping Lemma for Regular Languages] *Let L be any infinite regular language over an alphabet Σ with $|\Sigma| \geq 2$. There is a positive number m such that, for any number $n \geq 1$, any set $S \subseteq L \cap \Sigma^n$, and any series $(i_1, i_2, \ldots, i_k) \in ([1,n]_{\mathbb{Z}})^k$ with $\sum_{j=1}^{k} i_j \leq n$ for a certain number $k \in [1,n]_{\mathbb{Z}}$, the following statement holds. If $|S| > m$, then there exist two strings $x = x_1 x_2 \cdots x_{k+1}$ and $y = y_1 y_2 \cdots y_{k+1}$ in S with $|x_{k+1}| = |y_{k+1}|$ and $|x_{j'}| = |y_{j'}| = i_{j'}$ for each index $j' \in [1,k]_{\mathbb{Z}}$ such that, for every index $j \in [1,k]_{\mathbb{Z}}$, (i) $x \neq y$, (ii) $x_1 \cdots x_{j-1} y_j x_{j+1} \cdots x_{k+1} \in L$, and (iii) $y_1 \cdots y_{j-1} x_j y_{j+1} \cdots y_{k+1} \in L$.*

Proof. Note that, when $k = 1$, this lemma is indeed Lemma 3.1. Consider a dfa $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes L. Let $S \subseteq L \cap \Sigma^n$ satisfy $|S| > m$, where $m = |Q|^k$. To each string $s \in S$, we assign a k-tuple $(q_1, q_2, \ldots, q_k)$ of internal states of M such that, for each $j \in [1,k]_{\mathbb{Z}}$, M enters state q_j after scanning the $(\sum_{e=1}^{j} i_e)$th cell. There are at most $|Q|^k$ such tuples. Since $|S| > |Q|^k$, there are two distinct strings x, y in S such that they correspond to the same series of internal states, say $(q_1, q_2, \ldots, q_k)$. Write $x = x_1 x_2 \cdots x_{k+1}$ and $y = y_1 y_2 \cdots y_{k+1}$, where $|x_{k+1}| = |y_{k+1}|$ and $|x_{j'}| = |y_{j'}| = i_{j'}$ for every index $j' \in [1,k]_{\mathbb{Z}}$. Notice that, for each index $j \in [1,k]_{\mathbb{Z}}$, M enters the same internal state q_j after scanning x_j as well as y_j on the inputs x and y, respectively. Fix an index $j \in [1,k]_{\mathbb{Z}}$ arbitrarily. From the choice of x and y, we can swap the two blocks x_j and y_j in x and y without changing the acceptance condition of M. Therefore, the swapped strings $x_1 \cdots x_{j-1} y_j x_{j+1} \cdots x_{k+1}$ and $y_1 \cdots y_{j-1} x_j y_{j+1} \cdots y_{k+1}$ are both accepted by M. This gives the conclusion of the lemma. □

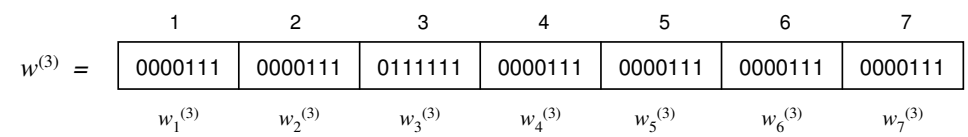

Figure 1: A string $w^{(j)}$ with $j = 3$ and $m = 7$

Let us demonstrate how to apply Lemma 3.4 to the deterministic context-free language GT.

Example 3.5 The deterministic context-free language GT is not in REG/n. Assuming that $GT \in$ REG/n, we choose an advice function h and a regular language L over an alphabet Σ such that, for every binary string x, $x \in GT$ iff $[\ {}^{x}_{h(|x|)}\ ] \in L$. Since L is an infinite regular language, we can apply Lemma 3.4 to L. Let m be a swapping-lemma constant. Without loss of generality, we can assume that m is odd and at least 3. Define $n = m^2$ and we are focused on the set $L \cap \Sigma^n$. Let $(i_1, i_2, \ldots, i_{m-1})$ be a unique series defined by $i_j = m$ for every index $j \in [1, m-1]_{\mathbb{Z}}$. This series makes each n bit-string partitioned into m blocks of equal size m. For each index $j \in [1, m]_{\mathbb{Z}}$, let $w^{(j)}$ denote the string $w_1^{(j)} w_2^{(j)} \ldots w_m^{(j)}$ of the following form: (i) the jth block $w_j^{(j)}$ equals 01^{m-1} and (ii) any other block $w_i^{(j)}$ equals $0^{m'+1}1^{m'}$, where $m' = \lfloor m/2 \rfloor$. Figure 1 gives an example of $w^{(j)}$ when $j = 3$ and $m = 7$. Since $\#_0(w^{(j)}) = \#_1(w^{(j)}) + 1$, this string $w^{(j)}$ belongs to GT. The desired set S is thus defined as $\{[\ {}^{w^{(1)}}_{h(n)}\ ], \ldots, [\ {}^{w^{(m)}}_{h(n)}\ ]\}$. Clearly, $|S| \geq m$. By Lemma 3.4, there are two distinct strings $w^{(k)}$ and $w^{(l)}$ ($k \neq l$) such that $[\ {}^{w^{(k)}}_{h(n)}\ ], [\ {}^{w^{(l)}}_{h(n)}\ ] \in S$ and $[\ {}^{\tilde{w}^{(k)}}_{h(n)}\ ], [\ {}^{\tilde{w}^{(l)}}_{h(n)}\ ] \in L$, where $\tilde{w}^{(k)}$ and $\tilde{w}^{(l)}$ are obtained respectively from $w^{(k)}$ and $w^{(l)}$ by swapping their lth blocks. Notice that, for each $i \in \{0, 1\}$, $\#_i(\tilde{w}^{(k)}) = \#_i(w^{(k)}) - \#_i(w_l^{(k)}) + \#_i(w_l^{(l)})$. Since $\#_0(w_l^{(k)}) = \#_1(w_l^{(k)}) + 1$, $\#_0(w_l^{(l)}) = 1$, and $\#_1(w_l^{(l)}) = m - 1$, it immediately follows that $\#_1(\tilde{w}^{(k)}) = \#_0(\tilde{w}^{(k)}) + m - 2$, which is greater than $\#_0(\tilde{w}^{(k)})$. This implies that $\tilde{w}^{(k)} \notin GT$, contradicting the conclusion of Lemma 3.4. Therefore, GT cannot be in REG/n. □

From Examples 3.3 and 3.5, since *Equal* and GT are both deterministic context-free, we can obtain a separation between DCFL and REG/n. This gives a strong separation between REG/n and DCFL/n, because REG/n = DCFL/n implies DCFL $\subseteq$ REG/n (using the fact that DCFL $\subseteq$ DCFL/n).

Proposition 3.6 DCFL $\not\subseteq$ REG/n. *Or equivalently,* REG/$n \neq$ DCFL/n.

4 Swapping Lemma for Context-Free Languages

We have shown the usefulness of our swapping lemma for regular languages by proving that three typical languages cannot belong to REG/n. Now, we turn our attention to CFL/n, the family of context-free languages with advice. A standard[†] pumping lemma for context-free languages helps us pin down, for example, the language $Dup = \{ww \mid w \in \{0,1\}^*\}$ (duplicating strings) into the outside of CFL. As in the case of regular languages, this pumping lemma is also of no use when we try to prove that Dup is not in CFL/n. This situation urges us to develop a new form of lemma, the *swapping lemma* for context-free languages.

To state our swapping lemma, we introduce the following notation for each fixed subset S of Σ^n. For any two indices $i, j \in [1, n]_{\mathbb{Z}}$ with $i + j \leq n$ and any string $u \in \Sigma^j$, the notation $S_{i,u}$ denotes the set $\{x \in S \mid u = midd_{i,i+j}(x)\}$. It thus follows that $S = \bigcup_{u \in \Sigma^j} S_{i,u}$ for each fixed index $j \in [1, n]_{\mathbb{Z}}$.

Lemma 4.1 [Swapping Lemma for Context-Free Languages] *Let L be any infinite context-free language over an alphabet Σ with $|\Sigma| \geq 2$. There is a positive number m that satisfies the following. Let n be any positive number at least 2, let S be any subset of $L \cap \Sigma^n$, and let $j_0, k \in [2, n]_{\mathbb{Z}}$ be any two indices satisfying that $k \geq 2j_0$ and $|S_{i,u}| < |S|/m(k - j_0 + 1)(n - j_0 + 1)$ for any index $i \in [1, n - j_0]_{\mathbb{Z}}$ and any string $u \in \Sigma^{j_0}$. There exist two indices $i \in [1, n]_{\mathbb{Z}}$ and $j \in [j_0, k]_{\mathbb{Z}}$ with $i + j \leq n$ and two strings $x = x_1x_2x_3$ and $y = y_1y_2y_3$ in S with $|x_1| = |y_1| = i$, $|x_2| = |y_2| = j$, and $|x_3| = |y_3|$ such that (i) $x_2 \neq y_2$, (ii) $x_1y_2x_3 \in L$, and (iii) $y_1x_2y_3 \in L$.*

The above form of our swapping lemma is similar to Lemma 3.4; however, we can no longer choose a pair (i, j) at our will. Moreover, the cardinality of a subset S must be much larger than that in the case of regular languages. Since the proof of this lemma is more involved than that of Lemma 3.4, we postpone it until the next section.

Meanwhile, we see how to use the swapping lemma for context-free languages. First, it is not difficult to show that Dup does not belong to CFL/n by applying the lemma directly.

[†]There are several well-known variants of pumping lemma for context-free languages. Ogden's lemma [7] is such a variant.

Example 4.2 The language Dup is not in CFL/n (and thus not in CFL). Let us assume that $Dup \in \mathrm{CFL}/n$ to lead to a contradiction. First, choose an advice function h and a context-free language L such that, for any binary string x, $x \in Dup$ iff $[\ {}^{x}_{h(|x|)}\ ] \in L$. Let m be a swapping-lemma constant for L. Second, choose any sufficiently large even number n satisfying that $2^{n/2} > 2mn^2$. Now, let us define a subset S as $S = \{[\ {}^{x}_{h(n)}\ ] \in L \mid |x| = n\}$. It suffices to satisfy the condition that $|S_{i,u}| \leq |S|/kmn$ for any index $i \in [1, n - j_0]_{\mathbb{Z}}$ and any string $u \in \Sigma^{j_0}$. Since $|S| = 2^{n/2}$ and $|S_{i,u}| = 2^{n/2-|u|}$ for any string $u \in \Sigma^{\leq n/2}$, we need to set $k = n/2$ and $j_0 = \lceil \log_2 mn^2 \rceil + 1$. By the swapping lemma for context-free languages, there are two indices $j \in [j_0, k]_{\mathbb{Z}}$ and $i \in [1, n-j]_{\mathbb{Z}}$ and two strings $x = x_1x_2x_3$ and $y = y_1y_2y_3$ in S with $|x_1| = |y_1| = i$, $|x_2| = |y_2| = j$, and $|x_3| = |y_3|$ such that (i) $x_2 \neq y_2$, (ii) $x_1y_2x_3 \in L$, and (iii) $y_1x_2y_3 \in L$. There are three cases to consider: (a) $i + j \leq n/2$, (b) $i < n/2 < i + j$, and (c) $n/2 \leq i$. Let us consider Case (a). Since $i \geq 1$ and $2 \leq j \leq n/2$, both x_2 and y_2 are respectively in the first half portion of x and y. Therefore, it is obvious that the swapped strings $x_1y_2x_3$ and $y_1x_2y_3$ are not of the form $[\ {}^{ww}_{h(n)}\ ]$. This is a contradiction. The other cases are similar. Therefore, Dup cannot be in CFL/n. □

In the above example, the choice of k is crucial. For instance, when $k = n/2 + 1$, there is a case where we cannot lead to any contradiction. Consider the following case. Take two strings $x = x_1x_2x_3$ and $y = y_1y_2y_3$ satisfying that $x_1 = y_1 = 0^{n/4-1}$, $x_3 = y_3 = 1^{n/4}$, and $x_2 = 0x_31x_10$, and $y_2 = 1y_31y_11$. Clearly, x and y are in $L \cap \{0,1\}^n$ and the swapped strings $x_1y_2x_3$ and $y_1x_2y_3$ are also in L.

Our next example is a slightly artificial language $Equal_6$, which consists of all strings w over the alphabet $\Sigma = \{a_1, a_2, \ldots, a_6, \#\}$ such that each symbol except # in Σ appears the same number of times in w; that is, $\#_a(w) = \#_b(w)$ for any pair $a, b \in \Sigma - \{\#\}$. Note that the complement $\overline{Equal_6}$ is in CFL. This containment is shown by considering an npda that behaves as follows: on input w, choose two distinct symbols, say a and b, in $\Sigma - \{\#\}$ nondeterministically and check if $\#_a(w) \neq \#_b(w)$. In other term, $Equal_6$ is in co-CFL. On the contrary, we can show that $Equal_6$ cannot belong to CFL/n.

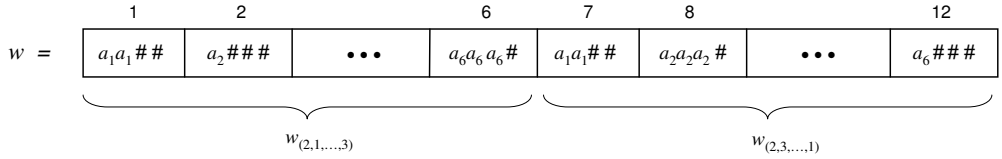

Figure 2: A form of w with $\#_{a_i}(w) = 4$ and $\#_{\#}(w) = 24$ when $n = 48$

Example 4.3 The language $Equal_6$ is not in CFL/n. Assuming that $Equal_6 \in \mathrm{CFL}/n$, we choose an advice function h and a language $L \in \mathrm{CFL}$ such that, for every string $x \in \Sigma^*$, $x \in Equal_6$ iff $[\ {}^{x}_{h(|x|)}\ ] \in L$. Since $L \in \mathrm{CFL}$, take a swapping-lemma constant m for L. Let $n = 864m$. For each symbol a_i in Σ, we use the notation $a_{(i,e)}$ for the string $(a_i)^e\#^{n/12-e}$ of length $n/12$. As a special case, we have $a_{(i,0)} = \#^{n/12}$ and $a_{(i,n/12)} = (a_i)^{n/12}$. For convenience, we denote by $w_{(e_1,e_2,\ldots,e_6)}$ the string made up by the following 6 blocks: $a_{(1,e_1)}a_{(2,e_2)}\cdots a_{(6,e_6)}$. Let S be the set consisting of all strings $[\ {}^{w}_{h(n)}\ ]$, where w is of the form $w_{(e_1,\ldots,e_6)}w_{(n/12-e_1,\ldots,n/12-e_6)}$ for any six indices $e_1, e_2, \ldots, e_6 \in [0, n/12]_{\mathbb{Z}}$. An example of such w is shown in Figure 2. Notice that, for any symbol $a \in \Sigma - \{\#\}$, if $w \in S$ then $\#_a(w) = n/12$. Moreover, $\#_{\#}(w) = n/2$. Now, we choose $j_0 = n/4$ and $k = n/2$. A simple observation gives $|S| = (n/12 + 1)^6$. Let u be an arbitrary string in Σ^{j_0}. To estimate $|S_{i,u}|$, we note that $|S_{i,u}| \leq |S_{1,\#^{n/4}}|$ for any index $i \in [1, n - j_0]_{\mathbb{Z}}$. This gives a simple upper bound: $|S_{i,u}| \leq (n/12 + 1)^3$. Obviously, since $n = 864m$, we have

$$|S_{i,u}| \cdot kmn \leq \left(\frac{n}{12} + 1\right)^3 \cdot \frac{mn^2}{2} = \frac{m(n+12)^5}{3456} < \left(\frac{n}{12} + 1\right)^6 = |S|.$$

The swapping lemma provides an index pair i, j with $n/4 \leq j \leq n/2$ and $i + j \leq n$ and a string pair $x, y \in \Sigma^n$ with $midd_{i,i+j}(x) \neq midd_{i,i+j}(y)$ such that $[\ {}^{x}_{h(n)}\ ], [\ {}^{y}_{h(n)}\ ] \in S$ and $[\ {}^{x'}_{h(n)}\ ], [\ {}^{y'}_{h(n)}\ ] \in L$, where x' and y' are two swapped strings defined as $x' = pref_i(x)midd_{i,i+j}(y)suf_{n-i-j}(x)$ and

$y' = pref_i(y)midd_{i,i+j}(x)suf_{n-i-j}(y)$. Since the substrings $midd_{i,i+j}(x)$ and $midd_{i,i+j}(y)$ have length j, $midd_{i,i+j}(x) \neq midd_{i,i+j}(y)$ implies that $\#_a(midd_{i,i+j}(x)) \neq \#_a(midd_{i,i+j}(y))$ for a certain symbol $a \in \Sigma - \{\#\}$. Hence, we conclude that $\#_a(x') \neq n/12$. This is a contradiction against the statement that $\left[\begin{smallmatrix} x' \\ h(n) \end{smallmatrix}\right] \in L$. Therefore, $Equal_6$ cannot be in CFL/n. □

Since the language $Equal_6$ belongs to co-CFL, as shown in Example 4.3, we can derive the following strong separation between CFL/n and co-CFL/n.

Proposition 4.4 co-CFL $\not\subseteq$ CFL/n. *Or equivalently,* co-CFL/$n \neq$ CFL/n.

The second part of the above proposition follows from the fact that co-CFL $\subseteq$ co-CFL/n. This proposition also indicates that CFL/n is not closed under complementation.

5 Proof of Lemma 4.1

As we have seen in Examples 4.2 and 4.3, the swapping lemma for context-free languages are a powerful tool in proving non-context-freeness with advice. This section will describe in detail the proof of Lemma 4.1. The proof requires an analysis of a stack's behavior of a nondeterministic pushdown automaton (or an npda).

5.1 Nondeterministic Pushdown Automata

Although there are several models to describe context-free languages, here, we use the machine model of npda's. We first review certain facts regarding the npda's. Let L be any infinite context-free language over an alphabet Σ with $|\Sigma| \geq 2$. Since Lemma 4.1 targets only inputs of length at least 2, it is harmless for us to assume that L contains no empty string λ. Now, consider a context-free grammar $G = (V, T, S, P)$ that generates L with $T = \Sigma$, where V is a set of variables, T is a set of terminal symbols, $S \in V$ is the start variable, and P is a set of productions. Without loss of generality, we can assume that G is in *Greibach normal form*; that is, P consists of the production rules of the form $A \to au$, where $A \in V$, $a \in \Sigma$, and $u \in V^*$. A process of transforming a context-free grammar into Greibach normal form is described in many undergraduate textbooks (*e.g.*, [4]).

Closely associated with the grammar G, we want to build an npda $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$, where Q is a set of internal states, Γ is a stack alphabet, δ is a transition function, $q_0 \in Q$ is the initial state, $z \in \Gamma$ is the stack start symbol, and $F \subseteq Q$ is a set of final states. For our npda M, we define $Q = \{q_0, q_1, q_f\}$, $z \notin V$, and $\Gamma = V \cup \{z\}$, $F = \{q_f\}$. To make our later argument simpler, we include two special *end-markers* ¢ and \$, which mark the left end and the right end of an input, respectively. Hereafter, we consider only inputs of the form ¢x\$, where $x \in \Sigma^*$, and we sometimes treat the endmarkers as an integrated part of the input. Notice that $|¢x\$| = |x| + 2$. For convenience, every tape cell is indexed with integers and the left endmarker ¢ is always written in the 0th cell. The input string x of length n is written in the cells indexed between 1 and n and the right endmarker \$ is written in the $n+1$st cell.

When we express the content of the stack of M as a series $s = s_1 s_2 s_3 \cdots s_m$, we understand that the leftmost symbol s_1 is located at the top of the stack and the s_m is at the bottom of the stack. At last, the transition function δ is defined as follows:

(1) $\delta(q_0, ¢, z) = \{(q_1, Sz)\}$;
(2) $\delta(q_1, a, A) = \{(q_1, u) \mid u \in V^*, P \text{ contains } A \to au\}$ for every $a \in \Sigma$ and $A \in V$; and
(3) $\delta(q_1, \$, z) = \{(q_f, z)\}$.

It is important to note that M is always in the internal state q_1 while the tape head scans any cell located between 1 and n. Note also that, during an accepting computation, the stack of the npda M never becomes empty because of the form of production rules in P. Therefore, we can demand that δ should satisfy the following requirement.

(4) For any symbol $a \in \Sigma$, $\delta(q_1, a, z) = \emptyset$.

Additionally, we modify the npda M and force its stack to increase in size by at most two by encoding several consecutive stack symbols (except for z) into one new stack symbol. For instance, provided that the original npda M increases its stack size by at most 3, we introduce a new stack alphabet Γ' consisting of (v_1), (v_1v_2), and $(v_1v_2v_3)$, where $v_1, v_2, v_3 \in \Gamma$. A new transition δ' is defined as follows. Initially, we define $\delta'(q_0, ¢, z') = \{(q_1, S'z')\}$, where $S' = (S)$ and $z' = (z)$. Consider the case where the top of a new stack contains a new stack symbol $(v_1v_2v_3)$, which indicates that the top three stack symbols of the original computation are $v_1v_2v_3$. If M applies a transition of the form $(q_1, w_1w_2w_3) \in \delta(q_1, a, v_1)$, then we instead apply $(q_1, (w_1w_2)(w_3v_2v_3)) \in \delta'(q_1, a, (v_1v_2v_3))$. In case of $(q_1, \lambda) \in \delta(q_1, a, v_1)$, we now apply $(q_1, (v_2v_3)) \in \delta'(q_1, a, (v_1v_2v_3))$. The other cases of δ' are similarly defined. See, *e.g.*, [4] for details.

Overall, we can assume the following extra condition.

(5) for any $a \in \Sigma$, any $v \in \Gamma$, and any $w \in \Gamma^*$, if $(q_1, w) \in \delta(q_1, a, v)$, then $|w| \leq 2$.

The aforementioned five conditions significantly simplify the proof of Lemma 4.1. In the rest of this paper, we assume that our ndpa M satisfies these conditions. For each string $x \in S$, we write $ACC(x)$ for the set of all accepting computation paths of M on the input x. Moreover, let $ACC_n = \bigcup_{x \in S} ACC(x)$.

5.2 Stack Transitions, Intervals, and Heights

For the proof of Lemma 4.1, we wish to present our key lemma, Lemma 5.1. To describe our lemma, we need to introduce several necessary notions and notations. An *intercell boundary* i refers to a boundary or a border between two adjacent cells—the ith cell and the $i+1$st cell—in our npda's input tape. We sometimes call the intercell boundary -1 the *initial intercell boundary* and the intercell boundary $n+1$ the *final intercell boundary*. Meanwhile, we fix a subset $S \subseteq L \cap \Sigma^n$, a string x in S, and a computation path p of M in $ACC(x)$. Along this path p, we assign to intercell boundary i a stack content produced after scanning the ith cell and before scanning the $i+1$st cell. For convenience, such a stack content is referred to as the "stack content at intercell boundary i." For instance, the stack contents at the initial and final intercell boundaries are both z, independent of the choice of accepting paths. Figure 3 illustrates intercell boundaries and a transition of stack contents.

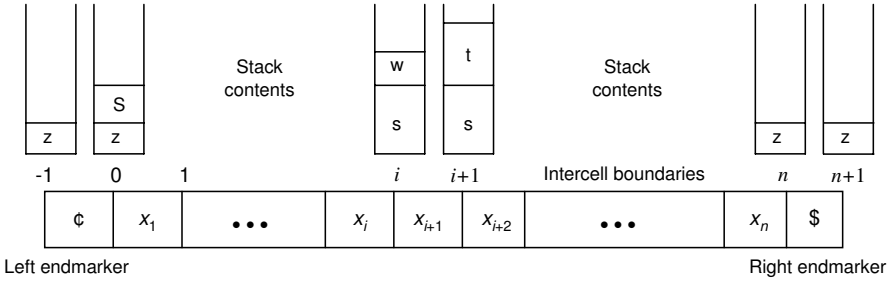

Figure 3: An example of intercell boundaries and stack contents

An accepting computation path of the npda generates a length-$(n+2)$ series $(s_{-1}, s_0, s_1, \ldots, s_n, s_{n+1})$ of stack contents with $s_{-1} = s_n = s_{n+1} = z$ and $s_0 = Sz$. We refer to this series as a *stack transition* associated with the interval $I_0 = [-1, n+1]_{\mathbb{Z}}$. More generally, when $I = [i_0, i_1]_{\mathbb{Z}}$ is a subinterval of I_0, we call an associated subsequence $\gamma = (s_{i_0}, s_{i_0+1}, \ldots, s_{i_1})$ a *stack transition* with the interval I. We define the *height* at intercell boundary b of γ to be the length $|s_b|$ of the stack content s_b at b. By our choice of the ndpa M given in Section 5.1, the minimal height is 1.

For our purpose, we hereafter focus our attention only on stack transitions γ with intervals $[i_0, i_1]_{\mathbb{Z}}$ in which (i) we have the same height ℓ at both of the intercell boundaries i_0 and i_1 and (ii) all heights within this interval are more than or equal to ℓ. We briefly call such γ *ideal*.

Let $I = [i_0, i_1]_{\mathbb{Z}}$ be any subinterval of I_0 and let $\gamma = (s_{i_0}, s_{i_0+1}, \ldots, s_{i_1})$ be any ideal stack transition with this interval I. For each possible height ℓ, we define the *minimal width*, denoted $minwid_I(\ell)$ (resp., the *maximal width*, denoted $maxwid_I(\ell)$), to be the minimal value (resp., maximal value) $|I'| = i'_1 - i'_0$ for which

(i) $I' = [i'_0, i'_1]_{\mathbb{Z}} \subseteq I$, (ii) γ has height ℓ at both intercell boundaries i'_0 and i'_1, and (iii) at no intercell boundary $i \in I'$, γ has height less than ℓ. Such a pair (i'_0, i'_1) produces a subsequence $\gamma' = (s_{i'_0}, s_{i'_0+1}, \ldots, s_{i'_1})$ of γ. In such a case, we say that γ' *realizes* the minimal width $minwid_I(\ell)$ (resp., maximal width $maxwid_I(\ell)$).

We say that a stack transition γ has a *peak at* i if $|s_{i-1}| < |s_i|$ and $|s_{i+1}| < |s_i|$. Moreover, γ has a *flat peak in* (i'_0, i'_1) if $|s_{i'_0-1}| < |s_{i'_0}| = |s_{i'_0+1}| = \cdots = |s_{i'_1}|$ and $|s_{i'_1+1}| < |s_{i'_1}|$. On the contrary, we say that γ has a *base at* i if $|s_{i-1}| > |s_i|$ and $|s_{i+1}| > |s_i|$; γ has a *flat base in* (i'_0, i'_1) if $|s_{i'_0-1}| > |s_{i'_0}| = |s_{i'_0+1}| = \cdots = |s_{i'_1}|$ and $|s_{i'_1+1}| > |s_{i'_1}|$. See Figure 4 for an example of (flat) peaks and (flat) bases.

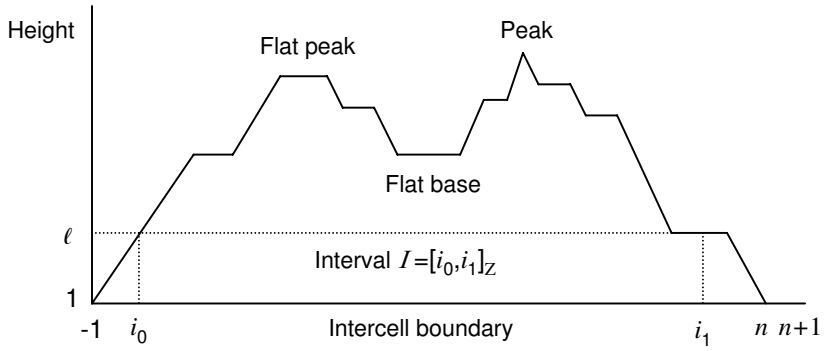

Figure 4: An example of track transition with interval $I = [i_0, i_1]_{\mathbb{Z}}$ and height ℓ

At last, we state our key lemma, which holds for any accepting computation path p without any assumption other than $j_0 \geq 2$ and $2j_0 \leq k \leq n$.

Lemma 5.1 [key lemma] *Let $S \subseteq L \cap \Sigma^n$, let x be any string in S, and let p be any computation path of M in $ACC(x)$. Assume that $j_0 \geq 2$ and $2j_0 \leq k \leq n$. For any interval $I = [i_0, i_1]_{\mathbb{Z}} \subseteq [-1, n+1]_{\mathbb{Z}}$ with $|I| > k$ and for any ideal stack transition γ with the interval I having height ℓ_0 at the two intercell boundaries i_0 and i_1, there are a subinterval $I' = [i'_0, i'_1]_{\mathbb{Z}}$ of I and a height $\ell \in [1, n]_{\mathbb{Z}}$ such that γ has height ℓ at both intercell boundaries i'_0 and i'_1, $j_0 \leq |I'| \leq k$, and $minwid_I(\ell) \leq |I'| \leq maxwid_I(\ell)$.*

Proof. Fix six parameters $(x, p, \gamma, \ell_0, I)$ given in the premise of the lemma. We prove the lemma by induction on the number of peaks or flat peaks along the computation path p of M in $ACC(x)$.

(Basis Case) In this particular case, there is either one peak or one flat peak in γ in the interval $I = [i_0, i_1]_{\mathbb{Z}}$. First, we consider the case where there is a peak. Let ℓ_1 be the height of such a peak. Note that $minwid_I(\ell) = maxwid_I(\ell + 1) + 2$ for any height ℓ with $\ell_0 \leq \ell < \ell_1$, because of the condition 5 provided for the npda's transition function δ. Now, let ℓ' be the maximal height satisfying that $minwid_I(\ell' + 1) \leq j_0 < minwid_I(\ell')$. Such ℓ' exists because $|I| > k > j_0$. Let $I_{min} = [i'_0, i'_1]_{\mathbb{Z}}$ be the *minimal* interval such that γ has height $\ell' + 1$ at the two intercell boundaries i'_0 and i'_1. Similarly, let $I_{max} = [i''_0, i''_1]_{\mathbb{Z}}$ be the *maximal* interval that satisfies a similar condition with $\ell' + 1$, i''_0, and i''_1. If $j_0 = minwid_I(\ell' + 1)$, then we choose the desired interval $I' = I_{min}$ and the height $\ell = \ell' + 1$ for the lemma. If $j_0 \leq maxwid_I(\ell' + 1)$, then we pick an interval I' satisfying that $I_{min} \subseteq I' \subseteq I_{max}$ and $|I'| = j_0$. We also define $\ell = \ell' + 1$ for the lemma. The remaining case to consider is that $maxwid_I(\ell' + 1) < j_0 < minwid_I(\ell')$. In this case, it follows that

$$j_0 < minwid_I(\ell') = maxwid_I(\ell' + 1) + 2 < j_0 + 2 \leq 2j_0 \leq k$$

since $j_0 \geq 2$. Let $I'_{min} = [\hat{i}_0, \hat{i}_1]_{\mathbb{Z}}$ be the minimal interval such that γ has height ℓ' at the two intercell boundaries $\hat{i}_0$ and $\hat{i}_1$. It is thus enough to define $I' = I'_{min}$ and $\ell = \ell'$ for the lemma.

Next, we consider the case where there is a flat peak in (i_2, i_3) with height ℓ_1. If $i_3 - i_2 \geq j_0$, then we choose $I' = [i_2, i_2 + j_0]_{\mathbb{Z}}$ and $\ell = \ell_1$ for the lemma. The other case where $i_3 - i_2 < j_0$ is similar to the "peak" case discussed above.

(Induction Step) First, let $c > 1$ and consider the case where there are c peaks and/or flat peaks in the given interval $I = [i_0, i_1]_{\mathbb{Z}}$ with $|I| > k$. Choose the lowest base or flat base within this interval. If we have

more than one such base and/or flat base, then we always choose the leftmost one. Now, consider the case where there is the lowest base at i_2 and let ℓ_2 be the height at i_2. Since γ is an ideal stack transition, we have $\ell_2 \geq \ell_0$. Let $I^* = [i'_0, i'_1]_{\mathbb{Z}}$ be the *largest* interval for which the height at both i'_0 and i'_1 equals ℓ_2. The choice of I^* implies that $|I^*| = maxwid_I(\ell_2)$. We split I^* into two subintervals $I_1 = [i'_0, i_2]_{\mathbb{Z}}$ and $I_2 = [i_2, i'_1]_{\mathbb{Z}}$. If $j_0 \leq |I^*| \leq k$, then we set $I' = I^*$ and $\ell = \ell_2$ for the lemma. If $|I^*| < j_0$, then a similar argument used for the basis case proves the lemma. Now, assume that $|I^*| > k$. Since $k \geq 2j_0$, either one of of I_1 and I_2 has size more than j_0. We pick such an interval, say I_3. Let γ' be a unique subsequence of γ defined in the interval I_3. If $|I_3| \leq k$, then we choose $I' = I_3$ and $\ell = \ell_2$ for the lemma. Let us assume that $|I_3| > k$. By the choice of I_3, γ' is an ideal stack transition. Since γ' has fewer than c peaks and/or flat peaks, we can apply the induction hypothesis to obtain the lemma.

Next, we consider the other case where there is the lowest flat base in (i_2, i_3). We define $I^* = [i'_0, i'_1]_{\mathbb{Z}}$ as before. Unlike the previous "lowest base" case, we need to split I^* into three intervals $I_1 = [i'_0, i_2]_{\mathbb{Z}}$, $I_2 = [i_2, i_3]_{\mathbb{Z}}$, and $I_3 = [i_3, i'_1]_{\mathbb{Z}}$. If either $|I^*| < j_0$ or $j_0 \leq |I^*| \leq k$, then it suffices to apply a similar argument used for the "lowest base" case. Now, assume that $|I^*| > k$. Since $k \geq 2j_0$, either one of the two intervals $I_1 \cup I_2$ and I_3 has size more than j_0. We pick such an interval. The rest of our argument is similar to the one for the "lowest base" case. □

5.3 Technical Tools

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ be our npda for L defined in Section 5.1. We have already seen fundamental properties of our npda in Section 5.2. Now, let us begin proving Lemma 4.1 by contradiction. First, we set our swapping-lemma constant m to be $|\Gamma|^2$ and assume that the conclusion of Lemma 4.1 is false for this m; that is, the following assumption (a) holds for four fixed parameters (n, j_0, k, S) given in the premise of the lemma. We fix these parameters throughout this subsection and its subsequent subsection.

(a) There are no indices $i \in [1, n]_{\mathbb{Z}}$ and $j \in [j_0, k]_{\mathbb{Z}}$ with $i + j \leq n$ and no strings $x = x_1x_2x_3$ and $y = y_1y_2y_3$ in S with $|x_1| = |y_1| = i$, $|x_2| = |y_2| = j$, and $|x_3| = |y_3|$ such that (i) $x_2 \neq y_2$, (ii) $x_1y_2x_3 \in L$, and (iii) $y_1x_2y_3 \in L$.

Recall that S is a fixed subset of $L \cap \Sigma^n$. Meanwhile, we fix additional five parameters $x \in S$, $j \in [j_0, k]_{\mathbb{Z}}$, $i \in [1, n-j]_{\mathbb{Z}}$, $v \in \Gamma$, and $p \in ACC(x)$. As a technical tool, we introduce the notation $G_{i,j,p}(x : v)$. Roughly speaking, $G_{i,j,p}(x : v)$ expresses a part of stack content that is newly produced from its original content vs during the head's scanning the cells indexed between $i+1$ and $i+j$, provided that the npda scans no symbol in s. Note that, when the npda is deterministic, the information on p can be discarded, because p is completely determined from x. More precisely, $G_{i,j,p}(x : v)$ denotes a unique string $t \in \Gamma^*$ (if any) that satisfies the following three conditions, along the computation path p with the input x.

1. The stack consists of vs at the intercell boundary i, where $s \in \Gamma^*$.
2. At the intercell boundary $i + j$, the stack consists of ts.
3. While the head scans any cell indexed between $i + 1$ and $i + j$, the npda never accesses any symbol in s; that is, no transition of the form $(q_1, w) \in \delta(q_1, a, r)$, where r is a symbol in s, is applied.

With the fixed parameters (i, j, v, t, p) described above, we use the notation $T^{(i)}_{j,v,t,p}$ to denote the set $\{x \in S \mid G_{i,j,p}(x : v) = t\}$. A crucial property of $T^{(i)}_{j,v,t,p}$ is stated in the following lemma.

Lemma 5.2 *We fix $j \in [j_0, k]_{\mathbb{Z}}$, $i \in [1, n-j]_{\mathbb{Z}}$, $p, p' \in ACC_n$, $v \in \Gamma$, and $t \in \Gamma^*$. For any two strings x, y in S, if $x \in T^{(i)}_{j,v,t,p}$ and $y \in T^{(i)}_{j,v,t,p'}$, then two swapped strings $pref_i(x)midd_{i,i+j}(y)suf_{n-i-j}(x)$ and $pref_i(y)midd_{i,i+j}(x)suf_{n-i-j}(y)$ are both in L.*

Proof. Assume that the npda's stack consists of vs (resp., vs') at an intercell boundary i along an accepting path p (resp., p') on an input x (resp., y). Since $x \in T^{(i)}_{j,v,t,p}$ (resp., $y \in T^{(i)}_{j,v,t,p'}$), the npda generates a stack content ts (resp., ts') at the intercell boundary $i + j$. Note that, during the head's scanning the cells indexed between $i + 1$ and $i + j$, the npda never accesses s (resp., s') along the path p (resp., p'). Therefore, we can swap two substrings $midd_{i,i+j}(x)$ and $midd_{i,i+j}(y)$ written in the cells indexed between $i+1$ and $i+j$ in x and y, respectively, without changing the acceptance condition of the npda. Therefore, the npda accepts the two

strings $pref_i(x)midd_{i,i+j}(y)suf_{n-i-j}(x)$ and $pref_i(y)midd_{i,i+j}(x)suf_{n-i-j}(y)$. This implies the conclusion of the lemma. □

Recall from Section 4 the notation $S_{i,w}$. Now, we consider the following statement.

(b) There are no $j \in [j_0, k]_{\mathbb{Z}}$, $i \in [1, n-j]_{\mathbb{Z}}$, $p, p' \in ACC_n$, $t \in \Gamma^*$, or $u, w \in \Sigma^j$ such that $u \neq w$, $T^{(i)}_{j,v,t,p} \cap S_{i,u} \neq \emptyset$, and $T^{(i)}_{j,v,t,p'} \cap S_{i,w} \neq \emptyset$.

Lemma 5.3 *The statement (a) implies the statement (b).*

Proof. Assume the statement (a). To show the statement (b), let us assume on the contrary that the statement (b) does not hold. This means that certain parameters (i, j, p, p', t, u, w) satisfy the following conditions: $u \neq w$, $T^{(i)}_{j,v,t,p} \cap S_{i,u} \neq \emptyset$, and $T^{(i)}_{j,v,t,p'} \cap S_{i,w} \neq \emptyset$. Now, we take two strings $x \in T^{(i)}_{j,v,w,p} \cap S_{i,u}$ and $y \in T^{(i)}_{j,v,w,p'} \cap S_{i,w}$. Notice that $|u| = |w| = j$. Since $x \in S_{i,u}$ and $y \in S_{i,w}$, it follows that $u = midd_{i,i+j}(x)$ and $w = midd_{i,i+j}(y)$. Lemma 5.2 then implies that the swapped strings $x' = pref_i(x)midd_{i,i+j}(y)suf_{n-i-j}(x)$ and $y' = pref_i(y)midd_{i,i+j}(x)suf_{n-i-j}(y)$ are both in L. This contradicts the statement (a). Therefore, the statement (b) holds. □

From Lemmas 5.2 and 5.3, the choice of accepting paths for strings in S is of little importance. Hence, for later convenience, we write $T^{(i)}_{j,v,t}$ to denote the union $\bigcup_{p \in ACC_n} T^{(i)}_{j,v,t,p}$.

5.4 Closing Argument

Under our assumption that Lemma 4.1 is false, we want to lead to a desired contradiction, which immediately proves the lemma. To achieve our goal, we utilize Lemma 5.1 given in Section 5.2. Notice that, by Lemma 5.3, the statement (b) now holds. Recall also that three parameters (n, j_0, k, S) are fixed through our proof.

For our convenience, we write Δ for the index set $\{(i, j, v, w) \mid j \in [j_0, k]_{\mathbb{Z}}, i \in [1, n-j]_{\mathbb{Z}}, v, w \in \Gamma\}$. The cardinality of this set Δ is equal to

$$|\Delta| = (k - j_0)(n - j_0 + 1)|\Gamma|^2 = m(k - j_0 + 1)(n - j_0 + 1)$$

since $m = |\Gamma|^2$. We want to assign each string x in S to a certain element (i, j, v, w) in Δ. For this purpose, we first show the following lemma, which can be obtained immediately from Lemma 5.1.

Lemma 5.4 *Assume that the statement (a) holds. Let j_0 and k satisfy that $j_0 \geq 2$ and $2j_0 \leq k \leq n$. For any string $x \in S$, there is an element $(i, j, v, w) \in \Delta$ for which $x \in T^{(i)}_{j,v,w}$.*

Proof. Let x be any string in S and consider any computation path p of the npda M in $ACC(x)$. We denote by γ the stack transition of M with the interval $I_0 = [-1, n+1]_{\mathbb{Z}}$. Lemma 5.1 guarantees, in the stack transition γ, the existence of a subinterval $I' = [i'_0, i'_1]_{\mathbb{Z}}$ and a height ℓ satisfying that $j_0 \leq |I'| \leq k$, $minwid_I(\ell) \leq |I'| \leq maxwid_I(\ell)$, and γ has height ℓ at both of the intercell boundaries i'_0 and i'_1. For our desired index values i and j, we define them as $i = i'_0$ and $j = |I'|$ $(= i'_1 - i'_0)$. Now, let us consider the changes of stack contents while M's head scans through the interval I'. Let vs be the stack content at the intercell boundary i'_0 and let ws' be the stack content at i'_1, where $v, w \in \Gamma$ and $s, s' \in \Gamma^*$. Note that $\ell = |vs| = |ws'|$. Since all heights in I' are at least ℓ, within this interval I', M does not access any symbol in s; hence, we conclude that $s = s'$. This implies that $G_{i,j,p}(x : v)$ equals w. Therefore, x should be in $T^{(i)}_{j,v,w}$. □

Lemma 5.4 establishes a key association of strings in S with elements in Δ. Using this association, we introduce a map e from S to Δ. For each string x in S, assuming an appropriate lexicographic order for Δ, we denote by $e(x)$ the *minimal* element $(i, j, v, w) \in \Delta$ satisfying that $x \in T^{(i)}_{j,v,w}$. Notice that this minimality requirement makes $e(x)$ uniquely determined from x. With this map e, we define $A_{i,j,v,w}$ as the set $\{x \in S \mid e(x) = (i, j, v, w)\}$. Obviously, it follows that $A_{i,j,v,w} \subseteq T^{(i)}_{j,v,w}$.

Now, we claim the following property of the map e.

Claim 1 *There exist two strings $x, y \in S$ and also two strings $u, z \in \Sigma^j$ such that $u \neq z$, $x \in S_{i,u}$, $y \in S_{i,z}$, and $e(x) = e(y) = (i, j, v, t)$.*

If this claim is true, then we take four strings (x, y, u, z) given in the claim. Since $e(x) = e(y) = (i, j, w, t)$, we obtain $x, y \in A_{i,j,v,w}$. Since $A_{i,j,v,w} \subseteq T^{(i)}_{j,v,w}$, it follows that $x \in T^{(i)}_{j,v,w} \cap S_{i,u}$ and $y \in T^{(i)}_{j,v,w} \cap S_{i,z}$. This is obviously a contradiction against the assumption (b) and hence the assumption (a). Therefore, Lemma 4.1 should hold.

What remains undone is the verification of Claim 1. Let us prove the claim. Since $e(\cdot)$ is a map from S to Δ, there is a certain element $(i, j, v, w) \in \Delta$ satisfying that $|A_{i,j,v,w}| \geq |S|/|\Delta|$. Fix such an element (i, j, v, w). For any string $u \in \Sigma^j$, since $|\Delta| = m(k - j_0 + 1)(n - j_0 + 1)$, we obtain

$$|S_{i,u}| < \frac{|S|}{m(k - j_0 + 1)(n - j_0 + 1)} = \frac{|S|}{|\Delta|} \leq |A_{i,j,v,w}|,$$

where the first inequality is one of the premises of Lemma 4.1. Since $S = \bigcup_{u \in \Sigma^j} S_{i,u}$, the above inequality concludes that there are at least two distinct strings $u, z \in \Sigma^j$ for which certain strings $x \in S_{i,u}$ and $y \in S_{i,z}$ map to the same element (i, j, v, t). This completes the proof of the claim and therefore completes the proof of Lemma 4.1.

Acknowledgments. The author thanks Satoshi Okawa and Francois Le Gall for a helpful discussion on a fundamental structure of context-free languages.

References

[1] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase-structure grammars. *Z. Phonetik Sprachwisss. Kommunikationsforsch*, 14 (1961), 143–172.

[2] N. Chomsky. Three models for the description of language. *IRE Trans. on Information Theory*, 2(3), 113–124, 1956.

[3] C. Damm and M. Holzer. Automata that take advice. In *Proc. 20th Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, Vol.969, pp.149–152, Springer, 1995.

[4] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass., Addison-Wesley, 1979.

[5] R. M. Karp and R. Lipton. Turing machines that take advice. *L'Enseignement Mathematique*, 28 (1982), 191–209.

[6] K. Tadaki, T. Yamakami, and J. C. H. Lin. Theory of one tape linear time Turing machines. *Proc. 30th SOFSEM Conference on Current Trends in Theory and Practice of Computer Science*, Lecture Notes in Computer Science, Vol.2932, pp.335–348, Springer, 2004. See also arXiv:cs/0310046.

[7] W. Ogden. A helpful result for peoving inherent ambiguity. *Mathematical Systems Theory*, 2(3), 31–42, 1969.